QUESTION 1: [4 Marks]

Answer with **True** or **False** in the following:

1. (F) A socket may be connected to more than one host at a time and socket may not reconnect after it's closed.
2. (F) The java.net.Socket class allows you to create socket objects that perform all four fundamental socket operations: Connect to a remote machine, Send data, Receive data and Close the connection.
3. (F) The Socket()constructors do not just create a Socket object. They also attempt to connect the underlying socket to the remote server.
4. (T) A server socket binds to a randomly selected port on the local machine. Once it has successfully bound to a port, it listens for incoming connection attempts.
5. (T) When a server detects a connection attempt, it accepts the connection. This creates a socket between the client and the server over which the client and the server communicate.
6. (T) Multiple clients can connect to the same port on the server at the same time.
7. (T) Constructing a server socket is as simple as specifying the port you want to listen on, like this:

```
try {
        ServerSocket ss = new ServerSocket(80);
}
catch (IOException e) {
        System.err.println(e);
}
```

8. (T) When a ServerSocket object is created, it attempts to bind to the port on the local host given by the port argument. If another server socket is already listening to the port, then it waits until the connection closed and tries to bind again.

QUESTION 2: [5 Marks]

Multiple choice questions:

1. What is the name of the method used to start a thread execution?
   a. init();
   b. start();
   c. run();
   d. resume();

2. Which of the following will not directly cause a thread to stop?
   a. notify()
   b. wait()
   c. InputStream access
   d. sleep()

3. Which of the following will directly stop the execution of a Thread?
   a. wait()
   b. notify()
   c. notifyall()
   d. exits synchronized code

4. Which three guarantee that a thread will leave the running state?
   a. yield()
   b. wait()
   c. notify()
   d. notifyAll()
   e. sleep(1000)
   f. aLiveThread.join()
   g. Thread.killThread()

5. Assume the following method is properly synchronized and called from a thread A on an object B:
   wait(2000);
   After calling this method, when will the thread A become a candidate to get another turn at the CPU?
   a. After thread A is notified, or after two seconds.
   b. After the lock on B is released, or after two seconds.
   c. Two seconds after thread A is notified.
   d. Two seconds after lock B is released.

6. Which cannot directly cause a thread to stop executing?
   a. Calling the setPriority() method on a Thread object.
   b. Calling the wait() method on an object.
   c. Calling notify() method on an object.
   d. Calling read() method on an InputStream object.

7. Which method must be defined by a class implementing the java.lang.Runnableinterface?
   a. void run()
   b. public void run()
   c. public void start()
   d. void run(int priority)

8. Which will contain the body of the thread?
   a. run();
   b. start();
   c. stop();
   d. main();

9. Can we make the user thread as daemon thread if thread is started?
   a. Yes
   b. No

10. You designate a method as being synchronized by:
    a. Using the synchronized keyword
    b. Placing a synchronized section of code within it
    c. Naming the method synchronize

QUESTION 3: [6 Marks]

1.
```
public class MyRunnable implements Runnable
{
    public void run()
    {
    // some code here
    }
}
```

How can you create and start this thread? **[1 Mark]**

```
MyRunnable run = new MyRunnable();
Thread t = new Thread(run);
t.start();
```

2. What will be the output of the programs? **[5 Marks]**

a) **[1 Mark]**

```
class TestCallRun2 extends Thread{
    public void run(){
        for(int i=1;i<=5;i++){
            try{
                Thread.sleep(500);
                }
            catch(InterruptedException e){
                System.out.println(e);
                }
            System.out.println(i);
        }
    }
    public static void main(String args[]){
     TestCallRun2 t1=new TestCallRun2();
     TestCallRun2 t2=new TestCallRun2();

     t1.run();
     t2.run();
    }
}
```

1
2
3
4
5
1
2
3
4
5

because we are calling the method run()

b) What will be the output of the program? **[1 Mark]**

```
import java.lang. ;


public class Q126 implements java.lang.Runnable
{
    private int x;
    private int y;

    public static void main(String [] args)
    {
        Q126 that = new Q126();
        Thread t1 = new Thread (that);
        Thread t2 = new Thread (that);
        t1.setName("Thread t1"); t2.setName("Thread t2");
        t1.start(); t2.start();
    }
    public synchronized void run( ) /* Line 18 */
    {
        for (;;) /* Line 20 */
        {
            x++;
            y++;
            System.out.println(Thread.currentThread().getName()
                ": " "x = " + x + " y = " + y);

            try
            {
                // Sleep for five thousand milliseconds (5 sec),
                // to simulate work being done
                Thread.sleep(5000);
            }
            catch (InterruptedException ie) {}

        }
    }
}
```

Thread t1: x = 1 y = 1
Thread t1: x = 2 y = 2
Thread t1: x = 3 y = 3
:

And it will go to infinity, for only Thread (t1)

Q2 P1

c) What will be the output if we replace line 20 by: [1 Mark]

```
for (int i=0;i<5;i++) /* Line 20 */
```

```
Thread t1: x = 1 y = 1
Thread t1: x = 2 y = 2
Thread t1: x = 3 y = 3
Thread t1: x = 4 y = 4
Thread t1: x = 5 y = 5
Thread t2: x = 6 y = 6
Thread t2: x = 7 y = 7
Thread t2: x = 8 y = 8
Thread t2: x = 9 y = 9
Thread t2: x = 10 y = 10
Press any key to continue...
```

d) What will be the output if we remove the keyword *synchronized* in line 18? **[1 Mark]**

~~Thread t1: x = 0 y = 0~~
~~Thread t2: x = 0 y = 0~~
Thread t1: x = 1 y = 1
Thread t2: x = 1 y = 1
Thread t1: x = 2 y = 2
Thread t2: x = 2 y = 2
Thread t1: x = 3 y = 3
Thread t2: x = 3 x = 3

seque

And it will go to infinity in the same or different

e) What will be the output if we remove the keyword *synchronized* in line 18 and replace line 20 as in part c above? **[1 Mark]**

```
Thread t2: x = 2 y = 2
Thread t1: x = 2 y = 2
Thread t2: x = 4 y = 4
Thread t1: x = 4 y = 4
Thread t1: x = 5 y = 5
Thread t2: x = 5 y = 5
Thread t1: x = 6 y = 7
Thread t2: x = 6 y = 7
Thread t2: x = 8 y = 9
Thread t1: x = 8 y = 9
Press any key to continue...
```